VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvovaní individuální odborné praxe

Individual Professional Practice in the Company

2012

Tomáš Škorvan

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Škorvan**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: NetDirect s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

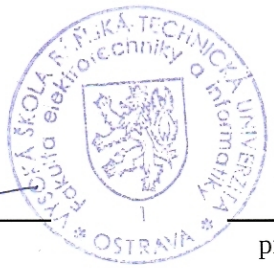
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Kožusznik, Ph.D.**

Konzultant bakalářské práce: Ing. Michal Rogozný

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 19. apríla 2012 .....

Chcel by som poďakovať Ing. Jan Kožuszníkovi, Ph.D. za pomoc pri tvorbe tejto práce a Ing. Michalovi Rogoznému za ponúknutu príležitosť praxe vo firme NetDirect s.r.o.

Abstrakt

Táto práca obsahuje riešenie problematiky prenosov dát pomocou SQL a XML, rozpoznávanie duplicitných záznamov, ich spracovanie a zmazanie. Popisuje prácu s XML v databáze, zmenu štruktúry XML pomocou XSLT šablón a C#. Zber dát z viacerých databáz internetových obchodov a tvorbu štatistík.

Klíčová slova: XML, XSLT, C#, Data transfer, Statistics, SQL, Database

Abstract

This thesis includes solving issues of data transmission using SQL and XML, identification of duplicate entries, their processing and deletion. Thesis describes work with XML in database, change in structure of XML by means of XSLT templates and C#. Data collection from several e-shops databases and creation of statistics.

Keywords: XML, XSLT, C#, Prenos dát, Štatistika, SQL, Databáza

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
XML	–	Extensible Markup Language
OLE DB	–	Object Linking and Embedding, Database
URL	–	Uniform Resource Locator
XSLT	–	Extensible Stylesheet Transformations

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Individuálnu odbornú prax som absolvoval vo firme NetDirect s.r.o. Firma sa zaoberá tvorbou e-shpov postavených na systéme FastCentrik a ShopCentrik, tvorbou web stránok postavených na systéme MediaCentrik. Prepojením e-shopov s ekonomickými, skladovými a logistickými ERP systémami. Pracoval som na oddelení FastCentrik ktoré implementuje e-shopy pre malých a stredných podnikateľov do 24 hodín. Zastával som pozíciu databázového špecialistu. Mojou úlohou bolo starať sa o import dát od dropshipingových partnerov, čiže prenos dát z XML do databáze, postupne sa moje úlohy rozšírili na celu databázu e-shopov , *ASP*, *C#*, *XSLT*, *HTML*, JavaScript.

2 Tvorba štatistík obratov e-shopov

Mojou prvou úlohou bolo vytvoriť štatistiku pre všetky e-shopy postavené na systéme FastCentrik a ShopCentrik a uložiť ich do jednej centrálnej tabuľky. Jednotlivé databázy e-shopov su rozdelené na viacerých serveroch. U systému FastCentrik nie je problem tvoriť štatistiku pretože su databázovo úplne totožné, kdežto u ShopCentrik sa historicky používalo niekoľko verzií. Pre ukladanie objednávok sa používali celkovo na oboch platformách tri druhy tabuliek. Ďalším problémom bolo, ako vložiť dáta z každého serveru do cieľovej databáze. Tento problém bol v podstate vyriešený pred mojim príchodom použitím linkovaného servera. Linkovaný server umožňuje spúšťať dotazy na vzdialenom servery pomocou OLE DB. OLE DB je API vytvorene Microsoftom pre prístup k rôznym dátovým zdrojom. Ďalším problémom je spustenie dotazu nad všetkými databázami a servermi. Microsoft SQL Server Management Studio umožnuje pomocou takzvaných Registered Servers spustiť príkaz nad skupinou serverov. Pre tvorbu dotazu prichádzalo v úvahu jedine dynamické SQL. Dynamické SQL umožňuje prispôsobiť si príkaz rôznim situaciam, keďže som potreboval spustiť príkaz vždy nad inou databázou potreboval som v SQL prikaze meniť názvy databáz. Názvy databáz sa nachádzaju na každom SQL servery v databáze *master* a tabulke *databases*. Pre spustenie nad každou databázou bol potrebný cyklus. Je mnoho spôsobov ako ho vytvoriť. Pomocou kurzora, alebo dočasná tabuľka s názvami databáz a cyklus while. Keďže kurzor mi nepripadal ako jedno z najrýchlejších riešení a dočasná tabuľka zbytočná kôli názvu databáze, zvoli som iný spôsob. Vytvoril som si premennú, nekonečný cyklus while a do premennej som si vždy uložil názov databáze nad ktorou sa bude vykonávať príkaz v danom cykle.

```
WHILE 1=1
BEGIN
   SELECT TOP 1 @db = name
   FROM master.sys.databases
   WHERE name > ISNULL(@db,'')
   ORDER BY name

   IF @@ROWCOUNT = 0 BREAK
END
```

Výpis 1: Cyklus

Zoradením tabuľky podľa mena a kontrolou na predchádzajúci názov uložený v premennej *@db* som vždy dostal ďalší názov databáze, pomocou *@@ROWCOUNT* som zistil či má cyklus ešte pokračovať alebo nie. Keďže príkaz bol vždy len nad jednou databázou, nepoužíval žiaden join s tabuľkou z inej databáze najjednoduchším spôsobom ako povedať že príkaz má byť spustený nad konkrétnou databázou bolo použitim *USE* v dynamickom SQL. *USE ['+@db+']* Ďalej už nasledovala len jednoduchá tvorba príkazu. Na FastCentriku je možnosť používať ako menu CZK tak aj EURO a ostatné. Pre štatisiky však musela byť mena rovnaká pre každý eshop čiže CZK. Pomocou príkazu with ktorý vráti dočasný pomenovaný výsledok príkazu som vybral data z tabuľky objednávok a k ním podľa meny aktuálny kurz. V ďalšej with klauzule som už len vytiahol dáta ktoré boli potrebné, priemernú objednávku, celkový obrat, počet objednávok a podobne. Ešte pred

cyklom som vytvoril dočasnú tabuľku do ktorej sa budú ukladať výsledky z každého cyklu while a na konci dynamického sql som len z with vybral všetky data. Názvy , poradie a datove typy stĺpcov z with aj dočasnej tabuľky sa zhodovaly takže som použil len *SELECT * FROM ...* Celé dynamické SQL som uložil do premennej a pomocou príkazu

```
INSERT INTO #tmp EXECUTE(@exec)
```

Výpis 2: Spustenie SQL a uloženie dát

som uložil výsledok z Dynamického SQL do dočasnej tabuľky.

Problému z rozdielnými názvami tabuľiek pre objednávky som vyriešil jednoducho, bolo potrebné vytvoriť tri variácie SQL príkazu pre každý typ tabuľky. Následne som spustenie dynamického SQL príkazu uzavrel do bloku *TRY CATCH* a pri chybe som spustil iný dotaz.

```
WHILE 1=1
BEGIN
    SET @exec = 'Prikaz1'
    begin try
        INSERT INTO #tmp EXECUTE(@exec)
    end try
    begin catch
        SET @exec = 'Prikaz2'
        begin try
        INSERT INTO #tmp EXECUTE(@exec)
        end try
        begin catch
            SET @exec = 'Prikaz3'
            begin try
                INSERT INTO #tmp EXECUTE(@exec)
            end try
            begin catch
                PRINT error_message()
            end catch
        end catch
    end catch
END
```

Výpis 3: Ošetrenie 3 typov tabuľiek

Takto som postupne otestoval a spustil dotaz na každom serveri, na každej databáze a na každej verzii e-shopu. Samozrejme pri výbere názvov databáz zo systémovej tabuľky som sa nevyhol spusteniu príkazov nad systémovými databázami, prípadne nad databázami ktoré su servisné alebo sa nepoužívaju priamo pre e-shop aby som mal túto chybu ošetrenú tak v poslednej sekcii som si vypísal chybovú správu. A už len ostavalo pomocou linkovaného serveru uložiť všetky dáta z jedného serveru do centrálnej tabuľky

```
INSERT INTO LNK_NETDIRECT.[Databaza].dbo.tabStatistiky select * from #tmp
```

Výpis 4: Použitie linkovaneho serveru

3 Optimalizácia odchytávania duplicít a chýb transportov

Databáza Fastcentriku pozná v podstate len tri druhy tansportov všetky prebiehajú pomocou komunikácie s XML a to

3.1 XML vo FastCentrik formáte z URL

Zahrňuje tabuľku v ktorej sú uložené url adresy odkazujúce na XML. XML musí byť vo fomáte aký podporuje Fastcentrik. Postupným prechádzaním záznamov s URL adresou sa spúšťa procedura s assembly. Assembly je napísaná v C#. Vstupom procedúry je url adresa a výstup xml. Výstup sa následne uloží do zdrojovej tabuľky, po vložení všetkých XML sa postupne spúšťajú procedúry ktoré dáta naimportujú.

3.2 XML v inom formáte z URL

Pre tento typ transportu je zvláštna procedúra ktorá ako jediná nie je na každom projekte rovnaká. Do premennej promocou procedúry volajúcej URL adresu sa vkladá XML do druhej premennej XSLT šablóna ktorá transformuje zdrojové XML do FastCentrik formátu a spustí sa ďalšia assembly ktorej výstupom je transformované XML ktoré sa ukladá do zdrojovej tabuľky s XML dátami a nasledne sa spúšťajú importné procedúry.

3.3 Ostatné importy

Ostatnými importmi sa myslí import z účtovných systémov ako Vario, MoneyS3, K2, Pohoda, Excel ručne cez administráciu, XML ručne cez administráciu. Pri týchto napojeniach sa XML vkláda priamo do tabuľky z nejakej asp stránky určenej pre import z daného sytému.

3.4 Rozoznanie duplicít

Importné procedúry pracujú pomocou príkazu *MERGE*, ktorý umožnuje vkladať alebo updatovať dáta z viacerých tabuliek do jednej. Jednou veľkou chybou u *MERGE* je že nieje možné updatovať jeden záznam viac ako raz. Pri tejto situácii *MERGE* skončí chybou a import neprebehne.

Odstraňovanie duplicit sa robí viacerými spôsobmi napr. v C# alebo v XSLT šablónach. Problém však je že každú s týchto operácii robil iný programátor a nie vždy úplne dokonale, čiže duplicita sa niekedy dostala do procedúry kde vyvolala chybu následne sa na konci procedúry nezmazali XML zo zdrojovej tabuľky a import ostal zaseknutý.

Na odchytávanie chýb mi prišlo najvhodnejšie použitie bloku *TRY CATCH*tento blok odchytí hocakú chybu . Túto chybu si uložím do premennej a pošlem do procedúry, ktorá vytvorí chybové XML, ktoré následne zapíšem do logovacej tabuľky. Z tejto tabuľky sa zobrazujú dáta v administrácií, presný popis chyby je možné z bezpečnostných dôvodov vidieť len po prihlásení pod servisným účtom.

Blok *TRY CATCH* síce odchytí chybu, ale bolo potrebné ešte odlíšit' duplicitný záznam a zmazat' ho.

Je viacero spôsobov ako mazat' duplicity pomocou klauzule *GROUP BY* odlíšit' duplicity a vložit' do dočasnej tabuľky, podľa ktorej neskôr zmažem s *TOP* duplicitné udaje a ponechám len jeden ktorý sa bude importovat'. Vzhľadom na to že na FastCentrik sú napojený aj dodávateľia ktorý poskytujú niekoľko sto tisíc kníh a ku knihám obrovské množstvo parametrov ktoré presahuje milión záznamov bolo nutné túto kontrolu spravit' čo možno najrýchlejšiu. Preto som sa rozhodol použit' *PARTITION BY* a funkciu *ROW_NUMBER()*. *PARTITION BY* slúži na zhlukovanie záznamov podľa určitích podobností je to obdoba *GROUP BY*. S použitím *ROW_NUMBER()* je možné vrátit' čísla riadkov v rámci špecifikovanej podmienky v *PARTITION BY*. Ako príklad uvediem parametre produktu.

1 produkt má N parametrov 1 parameter má N Hodnôt.

```
SELECT @sInternalMessage = @sInternalMessage
   +'sIdZbozi:  '+ CAST(sIdZbozi AS nvarchar(150))
   +'    , sJmenoParametru: '+sJmenoParametru
   +'    , sHodnotaParametru: '+sHodnotaParametru
   +'  #|VETADUP2BEFORE|# '+CAST(MAX(nDupRank) AS nvarchar(MAX))
   +'#|VETADUP2AFTER|#'+'<br />'
FROM
   (SELECT
      sIdZbozi,
      sJmenoParametru,
      sHodnotaParametru,
      nDupRank = ROW_NUMBER()OVER(PARTITION BY sIdZbozi,sJmenoParametru,
          sHodnotaParametru ORDER BY bVypnout)
   FROM #tmp) AS DUP
WHERE nDupRank > 1
GROUP BY sIdZbozi,sJmenoParametru,sHodnotaParametru
```

Výpis 5: Rozoznanie duplicit

Tento Select skonštruuje správu ktorá sa vypíše v administrácii. V stĺpci *nDupRank* je číslo riadku v sekcii ktorá je určená podľa pravidiel v *PARTITION BY*, čiže všetky produkty ktoré majú rovnaké *sIdZbozi*, *sJmenoParametru* a *sHodnotaParametru*. Predpokladajme že záznamy splňujúce túto podmienku sú tri v tom prípade každý s tých záznamov dostane číslo riadku 1 až 3, tým že ich zoradim podľa príznaku *bVypnout* bude na prvom mieste záznam ktorý sa má importovat'nie ten ktorý môže byt'vypnutý. Zo selectu však vyberiem iba záznamy ktoré majú *nDupRank* väčší ako 1, čiže duplicity ktoré chcem zmazat'. V prípade že sú 3 rovnaké záznamy select mi vyberie dva . V administrácii by však takáto správa nevyzerala príliš dobre pretože by tam bolo 2-krát to isté. Záznamy teda zhlukujem do jedného pomocou *GROUP BY* a uložím si ich do premennej *@sInternalMessage* v ktorej si poskladám vetu pre administráciu. V prípade že je záznamov viac tak si ukladám vety za seba a na koniec každej vkladám html znak *br* ktorý odriadkuje každu vetu.

Mazanie prebieha jednoducho, pomocou príkazu *DELETE* zmažem len tie záznamy ktoré majú *nDupRank* väčší ako 1.

```
DELETE DUP FROM
(SELECT
   sIdZbozi,
   sJmenoParametru,
   sHodnotaParametru,
   nDupRank = ROW_NUMBER() OVER(PARTITION BY sIdZbozi,sJmenoParametru,
        sHodnotaParametru ORDER BY bVypnout)
FROM #tmp) AS DUP
WHERE nDupRank > 1
```

Výpis 6: Zmazanie duplicit

Tento spôsob odstraňovania duplicít je veľmi rýchly a jednoduchý, navyše vďaka *ORDER BY* mi dáva čiastočnú kontrolu nad tým ktorý duplicitný záznam sa bude importovať a ktorý nie. Nakoniec sa premenná *@sInternalMessage* pošle do procedúry ktorá ju naformátuje do správnej podoby XML a vloží do logovacej tabuľky s príznakom, aby bola vyditeľná pre majiteľa e-shopu a ten si tak mohol duplicitu odstrániť sám. Pri správnom príkaze na výber duplicít už *MERGE* nebude hlásiť chybu, že updatuje jeden záznam viackrát.

4 Prevod dát z Demo verzie FastCentriku na ostrú verziu

Po spustení projektu Demo verzie FasCentriku čo je vlastne e-shop ktorý si môže každý potencionálny zákazník vyskúšať. Vznikol požiadavok na prevod užívateľských dát z Demo verzie na ostrý eshop. Časť práce prešla na administrátorov ktorý skopíruju užívateľské dáta z Demoverzie a naplnia nimi ostrý e-shop plus vytvoria zálohu databáze dema a vložia ju na server kde sa nachádza ostrý čistý projekt bez dát. Mojou úlohou bolo vytvoriť skrypt ktorý by dáta zo zálohy vložil na ostrý e-shop. Najjednoduchším spôsobom bolo vytvorenie XML vo formáte požadovanom pre transporty na FastCentrik a ich následne na importovanie. Keďže najlepší skrypt je taký ktorý funguje bez veľkých úprav na všetky prevody, bolo nutné ho spraviť univerzálne. Preto som sa rozhodol znovu použiť dynamické SQL. Na začiatku skryptu som si definoval premenné s názvom zdrojovej databázy z ktorej budem dáta čerpať a názvom cieľovej databázy kam sa budú dáta ukladať. Takže vytvorenie dotazu nebolo zložité. Správne pospájanie tabuliek, aby som dostal dáta, ktoré potrebujem a na konci klauzula s *FOR XML PATH ("zaznam"),ROOT("kategorie")* ktorá mi vytvorila xml. Každý riadok zo selectu uzavrela do elementu zaznam a root element celého xml bol s názvom kategorie. Celé toto XML som si uložil do premennej s dátovým typom XML a vložil do cieľovej databáze. Vysledný dotaz vyzeral takto

```
SET @SQL = N'
DECLARE @XML XML
   SELECT @XML=(

   SELECT
      Hodnota AS sIdKategorie,
      Hodnota AS sIdNadrizeneKategorie,
      Hodnota AS nCisloStromu,
      Hodnota AS sJmenoKategorie,
      Hodnota AS sPopisKategorie,
      Hodnota AS sJmenoObrazku,
      Hodnota AS bVypnout,
      Hodnota AS nPoradiZobrazeni
   FROM '+QUOTENAME(@SOURCEDB)+'.dbo.tabulkaKategorie
   FOR XML PATH (''zaznam''),ROOT(''kategorie'')
            )

IF @XML IS NOT NULL
INSERT INTO'+Quotename(@TARGETDB)+'.dbo.tblImportXmlSource(IDXML,xXmlFile)
VALUES(2,@XML)
'
```

Výpis 7: Vytvorenie XML

Po spustení všetkých príkazov, ktorými som vybral dáta a uložil ich do xml som už len jednoducho spustil transport. Čo však s dátami ktoré nie sú podporované transportmi ako napr. články, užívateľské nastavenia, nastavenia dopravy, natavenia platby, nastavenia užívateľských zámkov a pod. Pre tieto dáta som musel zvoliť prenos dat SQL to SQL . Problém však bol udržať väzby medzi tabuľkami. Preto som použil *SET IDENTITY_INSERT*

ON. Všetky primárne kľúče v databáze FastCentriku sú generované postupnou inkrementáciou ktorú zabezpečuje stĺpec s property *IDENTITY*, takže nie je možné vkladať primárny kľúč priamo cez *INSERT* je nutné najskôr nastaviť *IDENTITY_INSERT*, príkaz nám povolí vložiť primárny kľúč.

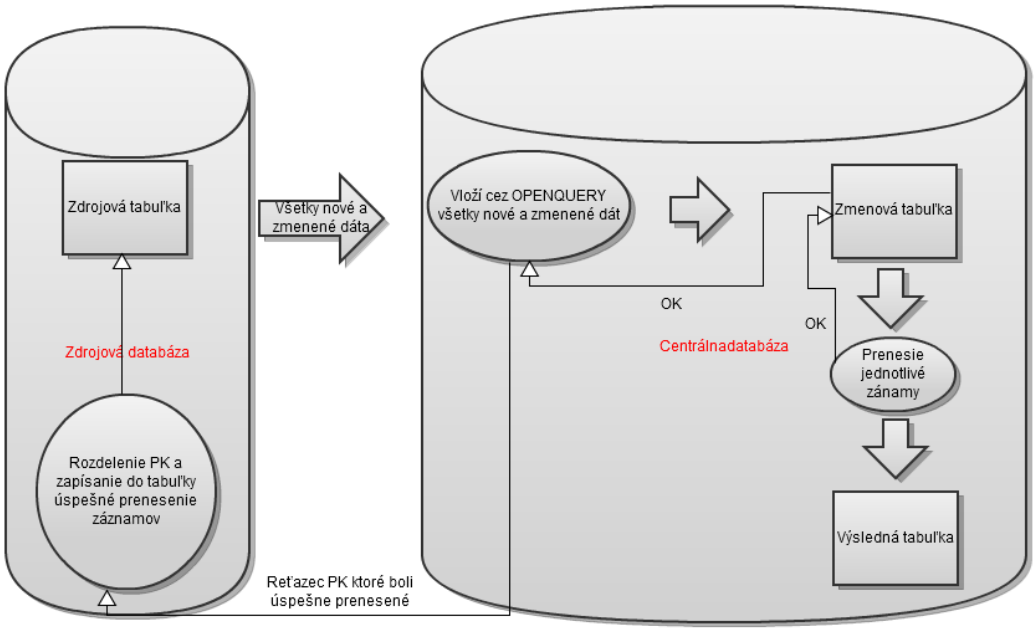
Postup popíšem na príklade prenosu zľavových kupónov. Kupóny reprezentujú dve tabuľky *Coupon* a *CouponCurrency*. Základné údaje o kupóne su v tabuľke *Coupon*, ako datum platnosti, kód, názov, popis. V tabuľke *CouponCurrency* je väzba na tabuľku *Coupon*, ďalej údaj o zľave buď percentá alebo cena a väzba na tabuľku *Currency* ktorá obsahuje udaje o mene. Najskôr treba vložiť údaje z tabuľky *Coupon* pretože tabuľka *CouponCurrency* obsahuje odkaz na túto tabuľku. Takže najskôr zmažem cieľovu tabuľku kupónov, potom zapnem *IDENTITY_INSERT* nad touto tabuľkou a vložím údaje nakoniec si vyberiem najväčší primárny kľúč a nastavím identity na jeho hodnotu aby ďalsie vloženie kupónu pri bežnom používani e-shopu neskončilo chybou, pretože PK by už v tabuľke bolo.

```
SELECT @PK = MAX(pkCoupon) FROM '+QUOTENAME(@TARGETDB)+'.dbo.Coupon
IF @PK IS NOT NULL
DBCC CHECKIDENT('"+QUOTENAME(@TARGETDB)+'.dbo.Coupon",RESEED,@PK)
```

Výpis 8: Zmena IDENTITY

Rovnako budem postupovať pri tabuľke *CouponCurrency* keďže zachovám primárne kľúče zachovám aj väzby medzi tabuľkami a pri ďalšom používaní nenastane chyba. Podobný prístup som uplatnil aj pri ostatných dátach ktoré nebolo možné preniesť transportmi. Výsledný skrypt má zatiaľ viac ako 1000 riadkov.

5 PEMIC-CENTRÁLA prenos SQL to SQL



Obrázek 1: Náčrt prenosu

Ďalšou mojou úlohou bolo po požiadavku od jedného veľkého dropshipingového partnera prenášať z ich databázy register autorov. Tento klient využíva špecialnu funkčnosť naprogramovanú priamo pre neho. Dáta sa prenášaju v pravidelných cykloch z ich databázy cez linkovaný server do našej databázy kde sa z nich tvoria XML a posielajú sa na všetky projekty ktoré používajú tieto dáta.

Pre túto úlohu som musel najskôr vytvoriť u nás kópiu tabuľky ktorú plnia na ich servery, táto tabuľka bola pre všetky dáta prenesené v jednom cykle Jobu. Do tejto tabuľky som pomocou procedúry a Linkovaného servera vložil všetky záznamy ktoré mali príznak *bSmazat = 0* to znamená že ešte neboli prenesené. Na spustenie príkazu som použil funkciu *OPENQUERY* prvý parameter tejto funkcie je názov linkovaného servera druhým je SQL príkaz.

Ďalej som si vytvoril kurzor ktorý vyberá primarný kľúč s prenesených dát a postupne som každý posielal do procedúry ktorá vybrala záznam z lokálnej tabuľky a vložila alebo zmenila záznam v ďalšej tabuľke. Táto tabuľka bola už výsledná a z nej sa čerpali záznamy pre tvorbu XML. Spätne som si označil záznam ako vložený resp. opravený. Ak nastala nejaká chyba a záznam sa nevložil alebo neopravil príznak prenesenia ostal na nule a pri ďalšom spustení Jobu sa pokúsil naimportovať znova. Záznamy v zdrojovej databáze

som musel rovnako označit' pri úspešnom prenesení, aby sa pri ďalšom spustení znova zbytočne neprenášali cez linkovaný server.

Na tento účel vznikla procedúra v zdrojovej databáze do ktorej som poslal primárne kľúče všetkých prenesených záznamov ako reťazec oddelený | procedúru som spúšťal rovnako cez linkovaný server. Tá si reťazec rozdelila a záznamy označila ako úspešne prenesené.

Tento spôsob som zvolil kôli historickým dôvodom keďže prenosy na produkty, kategórie a pod. boli už hotové, nemalo zmysel vytvárat' nový spôsob. Záznamy sa prenášali po jednom, z dôvodu lepšej kontroly chýb, prípadne kontrola na dĺžky reťazcov a pod.

Po prenesení dát do Centrálnej databázy bolo nutné vytvoriť XML, poslat' na každý projekt a Upadtovat' ho. Pre ukladanie vygenerovaných XML sa vytvorila nová databáza ktora obsahuje len jednu tabuľku do ktorej sa ukladajú XML pre všetky projekty. Projekty pre tohto dropshipingového partnera sú na samostatných serveroch ktoré sú prelinkované s centrálnou databázou, takže po vytvorení XML a uložení sa spustila procedúra, ktorá vybrala pre každy projekt XML ktoré bolo preňho určené. XML vložila ho do tabuľky na projekte a spustila import. Keďže na týchto projektoch je obrovský problém s veľkosťou posielaných dát je nutné aby sa importy spúšťali s oneskorením 5 minút.

5.1 PEMIC-CENTRÁLA menšie úlohy

Dropshipingový partner ktorý posielal dáta prevádzkuje aj vlastné e-shopy takže sa preňho upravovali a prispôsobovali rôzne menšie úpravy na importoch. Jednou z nich bolo prednostné poslanie nových popisov produktov, aby e-shop nebol vedený na porovnávačoch a vyhľadávačoch ako duplicitný. Každý z napojených e-shopov má svoje jedinečné identifikačné číslo, čiže s rozlíšením jedného na ktorý sa ma popis poslat' prednostne nebol problém. Bolo nutné však viest' časový záznam, kedy sa popis vytvoril. Keďže popisy produktov sú v inej tabuľke ako produkty ani s týmto problém nebol, lebo sa viedli záznamy kedy bola položka vložená. Takže už ostávalo len zakomponovat' pri tvorbe XML túto zmenu. Ak sa práve tvorí XML pre projekt ktorý je uprednostnený vloží sa popis, ak nie skontrolujem čas vloženia a podľa toho posielam popis, alebo nie.

```
CASE
   WHEN
   @sIdProvozniJednotky = '25899881_1_153_10630' OR
   @sIdProvozniJednotky = '25899881_1_153_10412'
   THEN ISNULL(P3.sKratkyPopis,'')
   ELSE CASE
         WHEN
         ISNULL(P3.dDatumVytvoreni,'1900-01-01 00:00:00.000')>DATEADD(HH,-48,
             GETDATE())
            THEN ''
         ELSE ISNULL(P3.sKratkyPopis,'')
         END
END AS sKratkyPopis,
```

Výpis 9: Riadenie posielania popisu

6 Napojenie na dodávateľa

Počas mojej praxe sa často stával s tym, že si e-shop objednal zákazník, ktorých chel napojit' dáta od nového dodávateľ. Niekedy sa mi však do rúk dostalo XML ktoré bolo určené pre porovnávače zboží ako napr. Zbozi.cz alebo Heureka.cz. Tieto XML mali jednu zásadnú chybu a to kategórie. Pretože XML bolo treba preniest'do formátu ktorý podporuje import Fastcentriku bolo nutné vytvorit'XSLT šablóny na prevod.

```
<kategorie>
  <zaznam>
    <sIdKategorie>2843</sIdKategorie>
    <nCisloStromu>1</nCisloStromu>
    <sIdNadrizeneKategorie/>
    <sJmenoKategorie>Kategorie 1</sJmenoKategorie>
  </zaznam>
</kategorie>
```

Výpis 10: Formát FastCentrik

```
<SHOP>
  <SHOPITEM>
    <PRODID>1</PRODID>
    <CATEGORY>Kola/Elektrokola</CATEGORY>
  </SHOPITEM>
  <SHOPITEM>
    <PRODID>1</PRODID>
    <CATEGORY>Rámy</CATEGORY>
  </SHOPITEM>
</SHOP>
```

Výpis 11: Poskytnutý Formát

Problém bol, ako získat'id kategorii a ich zaradenie pre tvorbu stromu kategorii. Najskôr však bolo nutné odstránit'duplicity, keďže v jednej kategorií je zaradených viac produktov. Použil som *xsl:variable* do ktorej som si uložil string kategórii oddelených nejakým oddeľovacím znakom. Variable sa plnila zavolaním *xsl:template*

```
<xsl:variable name="StoredCategories">
    <xsl:call-template name="StoreCategories"/>
</xsl:variable>
```

Výpis 12: Uloženie výstupu z Template

A template vyzerala takto

```
<xsl:template name="StoreCategories">
        <xsl:param name="delim" select="'|*|'"/>
        <xsl:param name="idx" select="1"/>
        <xsl:param name="buffer" select="''"/>

<xsl:variable name="curBuff">
<xsl:choose>
  <xsl:when test="not(contains(concat($delim,$buffer,$delim),concat($delim,/SHOP/SHOPITEM[$
      idx]/CATEGORY,$delim)))">
```

```
        <xsl:value-of select="$buffer"/>
        <xsl:if  test="string-length($buffer)&gt;0">
          <xsl:value-of select="$delim"/>
        </xsl:if>
        <xsl:value-of select="/SHOP/SHOPITEM[$idx]/CATEGORY"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$buffer"/>
      </xsl:otherwise>
    </xsl:choose>
    </xsl:variable>

    <xsl:choose>
      <xsl:when test="/SHOP/SHOPITEM[$idx+1]/CATEGORY">
        <xsl:call-template name="StoreCategories">
          <xsl:with-param name="idx" select="$idx+1"/>
          <xsl:with-param name="buffer" select="$curBuff"/>
         </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$curBuff"/>
      </xsl:otherwise>
    </xsl:choose>

    </xsl:template>
```

Výpis 13: Template na transformaciu kategórií

Je to v podstate cyklus ktorý prechádza každý element *CATEGORY* a v prípade že vstupný parameter buffer neobsahuje text z elementu *CATEGORY*, tak *CATEGORY* vloží do variable curBuff, ktorá sa použije pri ďalšej iterácii, kde vstupné parametre budu idx zvýšene o 1, aby sa vybral ďalší element *CATEGORY* a curBuff ako buffer aby sme zachovali už vytvorený string. Celé je to zabalené do podmienky ktorá v prípade že už nie je v XML ďalší element CATEGORY vráti curBuff s vytvoreným stringom bez duplicít. V prípade vzorového XML by string vyzeral takto *Kola/Elektrokola—*—Rámy*

Ďalším krokom je už vytvorenie kategórii a ich ID, keďže v Stringu stale nemám všetky potrebné kategórie tak je nutné ich ešte rozdelit' napr. u *Kola/Elektrokola* potrebujem vytvorit' dve kategórie a to *Kola* a *Elektrokola*. Pri vytváraní ID týchto kategórii som však nemohol použit' takto rozdelené stringy, pretože keď budem zaraďovat' produkty do kategorii tiež budem musiet' vytvárat' ID kategórie do ktorej sa produkt vloží. Produkt z id 1 je zaradený v kategórii Elektrokola, ale k dispozícii pre tvorbu ID kategorie je *Kola/Elektrokola* takže pre zaradenie použijem tento string a vytvorím ID. Strom kategórií a ich id ale vytvorím z názvu Kola čo je hlavná kategória a *Kola/Elektrokola* v tomto prípade sa mi uľahčí zaraďovanie produktov do kategórii, kde mi potom stačí vytvorit' z každého elementu *CATEGORY* nejake ID. Ako však dosiahnut' takéto rozdelenie? Použil som v šablóne *msxsl:script* s jazykom C#, jeho použitie je jednoduché definujem si element *msxsl:script* jeho atribúty budú *language="C#"implements-prefix="my"*, použitý jazyk ktorý nieje povinný a prefix ktorý budem používat' pri volaní metód zo skryptu tento povinný je. V sekcii *CDATA* budú napísane metódy. Ešte je však nutné ako atribúty stylesheet

zadať menný priestor *xmlns:msxsl="urn:schemas-microsoft-com:xslt"* namespace pre msxsl , *xmlns:my="urn:my-script"*. Predpona my je nutná pre určenie všetkých volaní funkcií definovaných vo vnútry *msxsl:script* [2]. Nakoniec ešte *exclude-result-prefixes="msxsl my"*, aby sa vo vystupnom XML neobjavovali deklarácie týchto namespacov . Takže použil som msxsl a v ňom metódu

```
public string getAllCategories(string inStr, string delim){

    System.Collections.Generic.List<string> list = new System.Collections.Generic.List<string>();
     string[] delims = new string[1] { delim };
    string[] items = inStr.Split(delims, System.StringSplitOptions.RemoveEmptyEntries);
    foreach (string item in items)
    {
        string subItem = item;
        if (!list.Contains(subItem)) { list.Add(subItem); }

        while (subItem.LastIndexOf("/") != -1)
        {
            subItem = subItem.Substring(0, subItem.LastIndexOf("/"));
             if (!list.Contains(subItem)) { list.Add(subItem); }
        }
    }
     list.Sort();
    return String.Join(delim, list.ToArray());
}
```

Výpis 14: Metoda v XSLT šablóne getAllCategories

Volanie metódy vyzeralo takto

```
<xsl:call-template name="CreateCategories">
    <xsl:with-param name="buffer" select="my:getAllCategories($StoredCategories,'|*|')"/>
</xsl:call-template>
```

Výpis 15: Spustenie C# scriptu

Ako vstup pre template *CreateCategories* sa použíl výstup z metódy *getAllCategories* ktorej vstupom je predchádzajúca vytvorená variable zo stringom kategorii bez duplicit a oddeľovač. V metode som si vytvoril List do ktorého som si ukladal výsledný string pre tvorbu kategórii. Do poľa stringov som vložil každú kategóriu, čiže u vzoroveho XML to boli dva prvky a to *Kola/Elektrokola* a *Rámy* a *System.StringSplitOptions.RemoveEmptyEntries* zaistí aby sa neukladali prázdne hodnoty. Potom je však ešte nutné rozdelit' *Kola/Elektrokola* takže cyklus foreach v ňom sa skontroluje či náhodou hodnota už v liste nieje, ak nieje vloží ju tam. Pri prvej iterácii vloží *Kola/Elektrokola* a v cykle While rozdeľuje na *Kola* a *Elektrokola*, keďže pre tvorbu id kategorie *Elektrokola* použijeme tento reťazec *Kola/Elektrokola* nieje nutné poslednú kategoriu zapisovať do listu. Vo výsledku máme tieto kategórie *Kola*, *Kola/Elektrokola* a *Rámy* metóda zotriedi list a vráti string všetkých kategórii ktoré sa použiju pre tvorbu stromu kategorii na e-shope oddelených oddeľovačom —*— . Nakoniec už len ostáva vytvorit' XML, o to sa postará template *CreateCategories*

```
<xsl:template name="CreateCategories">
  <xsl:param name="buffer" select="''"/>
  <xsl:param name="delim" select="'|*|'"/>
  <xsl:choose>
    <xsl:when test="contains($buffer,$delim)">
      <zaznam>
        <sIdKategorie>
          <xsl:value-of select="my:getHash(substring-before($buffer,$delim))"/>
        </sIdKategorie>
        <sJmenoKategorie>
          <xsl:value-of select="my:getCategoryName(substring-before($buffer,$delim))"/>
        </sJmenoKategorie>
         <sIdNadrizeneKategorie>
          <xsl:value-of select="my:getHash(my:getParentCategoryPath(substring-before($
              buffer,$delim)))"/>
        </sIdNadrizeneKategorie>
      </zaznam>
      <xsl:call-template name="CreateCategories">
        <xsl:with-param name="buffer">
          <xsl:value-of select="substring-after($buffer,$delim)"/>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <zaznam>
        <sIdKategorie>
          <xsl:value-of select="my:getHash($buffer)"/>
        </sIdKategorie>
        <sJmenoKategorie>
          <xsl:value-of select="my:getCategoryName($buffer)"/>
        </sJmenoKategorie>
        <sIdNadrizeneKategorie>
          <xsl:value-of select="my:getHash(my:getParentCategoryPath($buffer))"/>
        </sIdNadrizeneKategorie>
      </zaznam>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Výpis 16: Vytvorenie nového vystupného formátu

Template začína podmienkou že ak vstupný buffer obsahuje oddeľovač tak treba zobrať iba časť pred oddeľovačom, čiže prvú kategóriu. Ak oddeľovač neobsahuje tak to znamená, že buď je buffer prázdny alebo tam je len jedna kategoria, alebo sa vytvorili všetky kategorie a ostala len posledná. Takže v našom prípade buffer na začiatku obsahuje kategóriu *Kola*. Pomocou XPath funkcie substring-before ju oddelí od ostatných kategórii a pošle ako vstup do metódy getHash, ktorá z názvu kategórie vytvorí ID.

```
public string getHash(string str){
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
   if ( str  !=  "")
   {
      byte[]  hash = new                                                    System.
          Security.Cryptography.MD5CryptoServiceProvider().ComputeHash(System.Text.
          Encoding.UTF8.GetBytes(str));
      for  (int  i  = 0;  i  < hash.Length; i++)
      {
         sb.Append(hash[i].ToString("x2")) ;
      }
   }
   return sb.ToString() ;
}
```

Výpis 17: Metoda v XSLT šablóne getHash

Do byte array si uložím zašifrovanú hodnotu použijem triedu *MD5CryptoServiceProvider* a pomocou jej metódy ComputeHash vypočítam hash názvu kategórie, keďže metóda ako vstup používa Byte array tak treba pomocou *GetBytes* previesť string. Mám vytvorené ID kategórie, teraz potrebujeme jej názov v prípade kategórie *Kola* je to jednoduché ale čo v prípade *Kola/Elektrokola* ID tejto kategórie musíme vytvárať z *Kola/Elektrokola* ale jej názov musí byť Elektrokola. Použil som znova C# aj keď by to pravdepodobne šlo aj pomocou XSLT, ale C# mi prišiel ako jednoduchšie riešenie takže vytvoril som si metódu

```
public string getCategoryName(string path){
   if (path.Contains("/"))
   {
      string[]  names = path.Split("/".ToCharArray());
      return names[names.Length-1];
   }
   else
   {
      return path;
   }
}
```

Výpis 18: Metoda v XSLT šablóne getCategoryName

Vstup je *Kola/Elektrokola* ak string neobsahuje "/"tak vráti cely názov, ak áno rozdelí string do poľa a vráti poslednú vloženú hodnotu čiže *Elektrokola* . Mám ID kategórie mám jej názov ešte je nutné zistiť či je kategória rootová alebo je to podkategória. V prípade podkategórie je nutné zistiť ktorá kategória jej je nadradená a získať jej id. Keďže riešenie v C# mi prišlo elegantnejšie aby som nemusel znova vetvit'XSLT šablónu, tak som vytvoril metódu ktorá mi získa názov.

```
public string getParentCategoryPath(string path){
            if (path.LastIndexOf("/") != -1)
              {
                return path.Substring(0, path.LastIndexOf("/"));
              }
            else{
              return "";
            }
        }
```

Výpis 19: Metoda v XSLT šablóne getParentCategoryPath

Metoda je veľmi jednoduchá v prípade že string obsahuje "/"tak vráti názov od začiatku až po posledný výskyt znaku "/". V prípade *Kola/Elektrokola* je to *Kola* v prípade *Kola-/Elektrokola/kolesá* vráti *Kola/Elektrokola* a z vráteného názvu znova spravím ID pomocou metódy *getHash*

```
<sIdNadrizeneKategorie>
    <xsl:value-of select="my:getHash(my:getParentCategoryPath($buffer))"/>
</sIdNadrizeneKategorie>
```

Výpis 20: Vytvorenie ID nadradenej kategórie

Takže máme vytvorene XML stromu kategórii v našom formáte, ako som už spomínal tým, že sa vytvárali ID kategórií z celej cesty. Pre kategóriu *Elektrokola* je táto cesta *Kola-/Elektrokola*, tým sa veľmi zjednodušilo zaraďovanie produktov do kategórii. V šablóne ktorá vytvára XML pre zaraďovanie produktov do kategórii sa použíje len C# metóda *getHash* ktorá vráti ID kategórie z elementu *CATEGORY*.

7 Služba balík na poštu

Balík na poštu je služba českej pošty, kde si zákazník môže dať poslať balík na poštu ktorú si pri objednávke vyberie. Pre túto novu funkciu bolo nutné aby na každom servery bola dostupná tabuľka s poštami ktoré su aktuálne a s ich otváracími hodinami. Toto som mal zabezpečiť ja. Každý server má jednu databázu Monitor v ktorej sú uložené a synchronizované dáta ku ktorým má prístup každý projekt na serveri. Tieto dáta su pre každý projekt rovnaké takže nieje nutné aby boli uložene v každej databáze. Rovnako to bolo aj zo službou Balík na poštu tabuľky boli určene pre projekty len na čítanie. Prvé čo bolo nutné urobiť bolo zabezpečiť synchronizáciu z XML, ktoré poskytuje česká pošta do tabuľky na server ku ktorému sú všetky ostatné nalinkované. Dve tabuľky

```
CREATE TABLE [dbo].[tbl_SYNC_Posta](
    [pkTbl_SYNC_Posta] [int] IDENTITY(1,1) NOT NULL,
    [sPSC] [nvarchar](5) NOT NULL,
    [dLastUpdate] [datetime] NULL
)
CREATE TABLE [dbo].[tbl_SYNC_PostaOteviraciDoba](
    [pk_Tbl_SYNC_PostaOteviraciDoba] [int] IDENTITY(1,1) NOT NULL,
    [fkTbl_SYNC_Posta] [int] NOT NULL,
    [nDEN] [int] NOT NULL,
    [sOD] [nvarchar](40) NOT NULL,
    [sDO] [nvarchar](40) NOT NULL,
    [sDEN] [nvarchar](50) NULL,
    [dLastUpdate] [datetime] NULL
)
```

Výpis 21: Štruktúra tabuľiek

Tieto tabuľky sa plnili dátami z XML. Procedúra začala vytvorením dočasných tabuliek na uloženie dát z XML dočasné tabuľky mali rovnaké názvy stĺpcov ako elementy XML aby sa ľahšie pracovalo. Potom pomocou assembly stiahnem XML z url adresy a uložím si ho do premennej typu XML, aby som mohol dáta z XML uložiť do dočasných tabuľiek je treba pripravit si XML spustením *EXEC sp_xml_preparedocument @nXmlPrepare OUTPUT, @xXmlOut;*

Parameter *@nXmlPrepare* je integer výstupná hodnota z procedúry. Procedúra prečíta xml zo vstupu v tomto prípade premennú *xXmlOut* pomocou MSXML parsera a takto pripravený dokument poskytne na použitie v SQL. Tento dokument je stromovou reprezentáciou rôznych uzlov v XML dokumente ako elementov atribútov a podobne. Procedúra vracia integer hodnotu ktorá sa používa ako handle pre prístup k vnútornej reprezentácii XML dokumentu tento Handl je validny do doby pokiaľ ho nezrušíme a to pomocou procedúry *EXEC sp_xml_removedocument @nXmlPrepare;*

```
OPENXML( @nXmlPrepare, 'zv/row', 2)
    WITH
    (
        PSC nvarchar(5),
        NAZ_PROV nvarchar(50),
    ) ;
```

Výpis 22: Načítanie dát pomocou OPENXML

Prvým argumentom je výstupná premenná z predchádzajúcej procedúry teda handl určený pre prístup k XML . Druhým argumentom je XPath príkaz ktorý určí uzly XML, ktoré budú spracované ako riadky do tabuľky. Tretím parametrom je flag ktorý určuje mapovanie. Toto mapovanie bude použité na prepojenie dát z XML do relačnej sady riadkov. Tento parameter je voliteľný
Flag = 1 - Hovorí že načítane dáta XML majú hodnoty stĺpcov uložené vo vnorených atribútoch uzlu
Flag = 2 - Hovorí že načítane dáta XML majú hodnoty stĺpcov uložené ako samostatné vnorené elementy
Hodnoty súboru XML sú potom použité ako riadky vrátené z funkcie *OPENXML()*. Ďalej nasleduje schéma databázy do ktorej chceme výsledky uložit'[1].

Po vložení dát do dočasných tabuliek už nasledovalo len updatovanie alebo vloženie dát do tabuľky v databáze a to pomocou funkcie *MERGE* v tabuľkách bol vytvorený stĺpec dLastUpdate ktorý pri vložení alebo oprave riadku zmení svoju hodnotu na aktuálny dátum a čas. Na začiatku procedúry som si takisto vytvoril premennú @dLastUpdate do ktorej som si pomocou funkcie *GETDATE()* vložil aktuálny čas a dátum. Tieto časové údaje mi slúžili na konci pre mazanie dát z tabuliek. *DELETE FROM tbl_SYNC_Posta WHERE ISNULL(dLastUpdate,'') < @dLastUpdate* V prípade že čas v tabuľke, v stĺpci *dLastUpdate* je menší ako premenná *@dLastUpdate* znamená to že záznam nebol pri spustení procedúry ani upravený, ani novo vložený, teda sa už v XML nenachádza a nie je aktuálny takže je ho možné zmazat'. Dáta sú už v tabuľke, ešte ich treba synchronizovat'z ostatnými servermi. Na každom servery je job ktorý ma za úlohu každú hodinu synchronizovat'tabuľky ktoré sú už vytvorené spúšťa procedúru do ktorej som pridal svoj kód, aby sa synchronizovali aj tieto dve tabuľky. Synchronizačný kód sa skladal z troch častí pre každú tabuľku a to

```
DELETE FROM dbo.tblPosta
  WHERE pkTblPosta NOT IN (SELECT pkTbl_SYNC_Posta FROM LNK.[_Main].dbo.
      tbl_SYNC_Posta)
```

Výpis 23: Zmazanie záznamov ktoré už niesu v zdrojovej tabuľke

```
UPDATE T
  SET     sPSC = Source.sPSC,
        sNAZEV_PROV = Source.sNAZEV_PROV,
        dLastUpdate = Source.dLastUpdate

  FROM dbo.tblPosta T JOIN LNK.[_Main].dbo.tbl_SYNC_Posta Source ON T.pkTblPosta = Source
      .pkTbl_SYNC_Posta
```

Výpis 24: Opravenie položiek ktoré sa nachádzajú v oboch tabuľkách

```
INSERT INTO dbo.tblPosta (pkTblPosta, sPSC, sNAZEV_PROV)
SELECT pkTbl_SYNC_Posta, sPSC, sNAZEV_PROV
FROM LNK.[_Main].dbo.tbl_SYNC_Posta
WHERE pkTbl_SYNC_Posta NOT IN (SELECT pkTblPosta FROM dbo.tblPosta)
```

Výpis 25: Vloženie nových položiek

Každý s týchto príkazov bol ešte ošetrený chybou a to pomocou príkazu *IF @@ERROR<>0 GOTO OnFAIL*.

@@ERROR vráti kód chyby ak nejaká nastala v opačnom prípade je 0, ak nastane chyba pomocou príkazu GOTO sa presunie na návestie *OnFAIL: RAISERROR ('Eroor', 16, 1) RETURN* Kde pomocou Raiserror vyvolám vlastnú výnimku ako argumenty sú použité textový popis chyby, severity ktorá udáva závažnosť chyby a číslo od 1-127 je stavová hodnota obvykle je ale 1

8 Záver

Túto prax považujem za veľký úspech, keďže som nazbieral obrovské množstvo skúseností z praxe čo je pre študenta bez skúseností veľmi dôležité. Pretože pri nástupe u väčšiny firiem je prax v obore požadovaná. Nemenej dôležité je nazbieranie znalostí z rôznych programovacích jazykov a ich využitie v celku pre tvorbu komplexných programov a vytváranie jednoduchých, rýchlych a funkčných riešení problémov. Naučil som sa pracovať v týme, ako aj samostatne a naučil sa množstvo nových programovacích jazykov a techník.

Tomáš Škorvan

9 Reference

[1] PRICE, Jason. *C#: programování databází.* Praha: Grada, 2005, 623 s. ISBN 80-247-0982-1.

[2] ESPOSITO, Dino. *XML: efektivní programování pro .NET.* 1. vyd. Překlad Jaroslav Černý. Praha: Grada, 2004, 596 s. ISBN 80-247-0775-6.